M is Going Places

Clemens Szypersk
Development Manager, Transformations & Streaming
Information Services, Data Platform Group, C&E

TCN Mindswap
Programming Languages
March 3, 2014

Driving your own car, anyone?

Having a chauffeur was more than a luxury. It was a necessity. So many things could go wrong, requiring a technician's skills.

And it limited who could afford to own and use a car.

Self-Service Revolution

"The worldwide demand for cars will not exceed one million – even if just for a scarcity of available chauffeurs."

Gottlieb Daimler, Inventor, 1901

Sticking to the technology status quo ?

"There is no reason for any individual to have a computer in his home."

Ken Olson, Founder and CEO of Digital Equipment Corp, 1977 at Convention of the World Future Society

Programmer

A person skilled in designing and developing programs.

The chauffeur of your computer!

- Programmers write solutions (programs) in a programming language.
  - Requires intersection of programming skills (how?) and domain knowledge (what?)
- Programming languages themselves are the subject of a design activity.
  - Facts and opinions abound: usability, expressiveness, correctness by construction, readability vs. writability, simplicity, style, ...

Properties of Programming Languages

Tools need to match the problem space, the audience expected to use the tool, and the expectation space of the desired outcome when using the tool.

- Read-Only Languages
  - SQL (Structured Query Language) – many learn to read SQL, only a few can write non-trivial SQL
- Write-Only Languages
  - Pearl – many learn to write scripts, but most cannot even read what they wrote themselves a day ago
- Impedance Mismatch
  - "Ceremony" or lack of expressiveness force cumbersome formulations of solutions in a given problem domain
- Requirements Mismatch
  - Functionally good expressions end up failing expectations of performance, security, etc.

Law of the Instrument

"I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail."

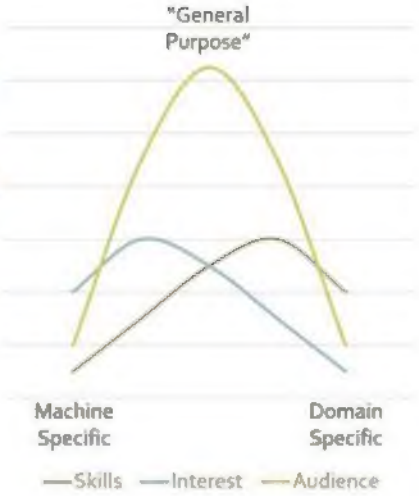
Abraham Maslow, Psychologist, 1966

Programming Languages

Given a computer with some primitive operations and a problem to solve.

Formulate a composition of instructions to the computer that solve the problem.

- Instructions can be very **low-level** (close to the machine's primitive operations)
- Instructions can be very **high-level** (close to the problem domain at hand)
- Most languages strike a **balance**
  - Too low-level (limited audience, limited target machines)
  - Too high-level (limited audience, limited problem domains)



Programmer

A person skilled in designing and developing programs.

The chauffeur of your computer!

- Why not "drive" your own computer to go where you want to go?
  - This is not about "using" a computer application, in the simple sense
- Why not write the programs you need to get your job done, yourself?
  - This is not about "programming" a computer either, in the fullest sense
- Why not master a programming language?
  - If the language is Abstract Algebra, you'll be in trouble; if it is Pidgin, you are in trouble too

Self-Service Programming

Think of cars that most people can learn to drive.

Clearly not to the limit of what "cars" can be; think 18-wheeler trucks or F1.

- Query by Example

  Moshé M. Zloof, IBM Research, mid-1970s

- Generalizes to Programming by Example
  - Using direct manipulation, change results of a program, causing the system to adjust that program.
- Users can watch the effect on the underlying program – and learn from that.
  - Some users pick up ways to change their programs directly, naturally learning the underlying programming language.
  - Requires uniform and simple languages.

Audience-Specific Programming Languages

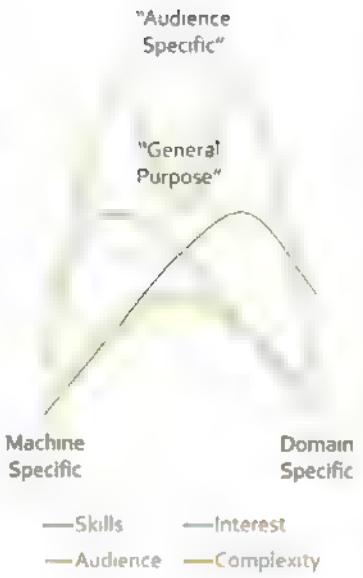
Consider a variety of *personas* that characterize

their tasks done

that an audience

- Languages that strive to be "general purpose" end up being quite right at most anything.
- To compensate, such languages develop a large specialized but overlapping capabilities.

  The ideal maximized audience is subdued by complexity.
- Larger audiences can be served with simpler languages to either side of the "general purpose" point

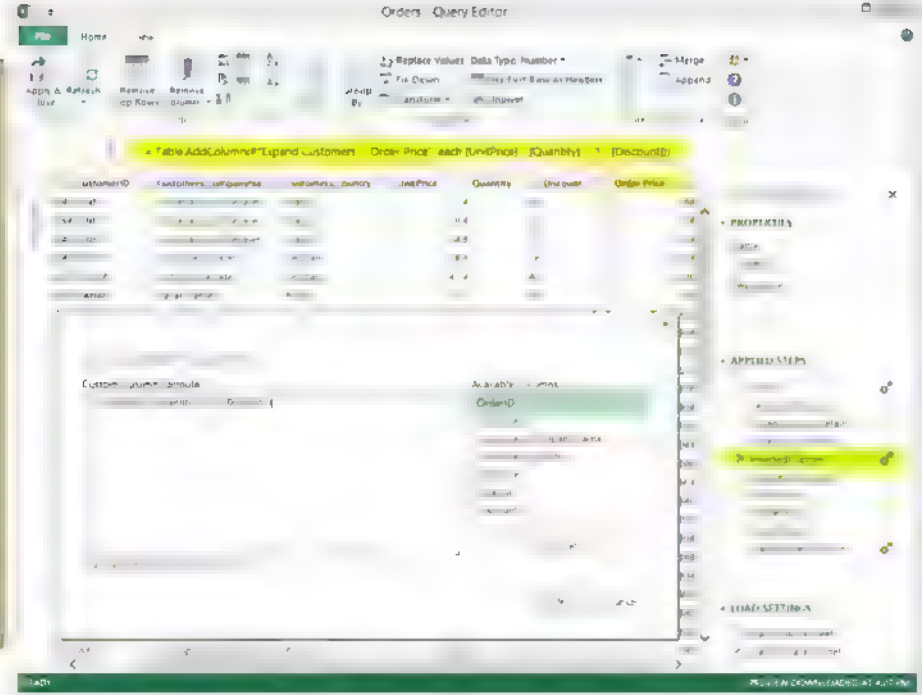


Anyone can drive a car

is doing until it's too late."

Anyone can

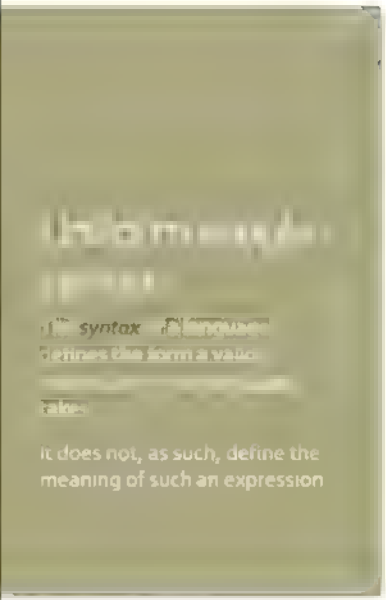
Power Query



"M" - a simple programming language

Language (often referred to "M" for short)

- [illegible] is advanced Information Workers (Analysts etc.), Data Stewarts
  - Specifically, top 10% (ish) of Excel users
  - Litmus test: benefits from today's Excel formulas
- For that audience, the language should be
  - [illegible], easy to remember
  - Easy to read and write, [illegible], little use of non-standard symbols
  - Powerful, [illegible] for advanced user
  - [illegible] of "data models" (relational, hierarchical, semi-structured, etc.)



T-SQL

```
SELECT Orders.OrderDate, Products.OrderID, Products.ProductSKU
FROM Products
INNER JOIN Orders ON Products.OrderID = Orders.OrderID
ORDER BY ProductSKU;
```

T-SQL

```
SELECT  Orders.OrderDate, Products.OrderID, Products.ProductSKU
FROM    Products
INNER JOIN  Orders  ON  Products.OrderID = Orders.OrderID
ORDER BY  ProductSKU ,
```

C# LINQ syntax

```
from p in Products
join o in Orders on p.OrderID equals o.OrderID
orderby p.ProductSKU
select new { o.OrderDate, p.OrderID, p.ProductSKU }
```

T-SQL

```
SELECT Orders.OrderDate, Products.OrderID, Products.ProductSKU
FROM Products
INNER JOIN Orders ON Products.OrderID = Orders.OrderID
ORDER BY ProductSKU ,
```

C# LINQ syntax

```
from p in Products
join o Orders on p.OrderID equals o.OrderID
p.ProductSKU
select { o.OrderDate, p.OrderID, p.ProductSKU }
```

C# LINQ pattern

```
Products
    .Orders,
    p => p.OrderID, o => o.OrderID,
    (p, o) => new { o.OrderDate, p.OrderID, p.ProductSKU } )
    .OrderBy( p => p.ProductSKU )
```

Uniform simple syntax

T SQL

```
SELECT  Orders.OrderDate, Products.OrderID, Products.ProductSKU
FROM    Products
INNER JOIN  Orders  ON  Products.OrderID = Orders.OrderID
ORDER BY  ProductSKU ,
```

C# LINQ syntax

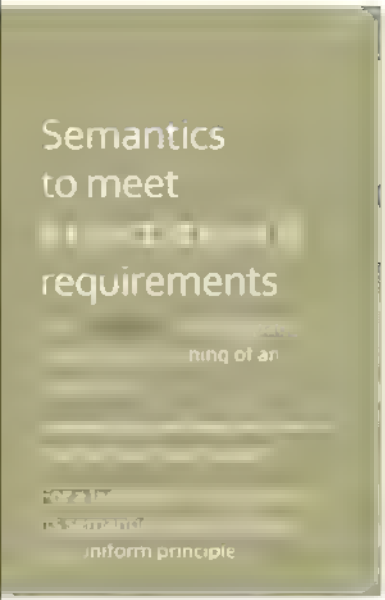
```
from p in Products
join o    Orders   p.OrderID ==    o.OrderID
select p.ProductSKU
select new { o.OrderDate, p.OrderID, p.ProductSKU }
```

C# LINQ pattern

```
Products
    .    (Orders,
        p   > p.OrderID, o   > o.OrderID,
        (p, o)  >   { o.OrderDate, p.OrderID, p.ProductSKU } )
    .       ( p   > p.ProductSKU )
```

"M"

```
let Joined  =              ( Products, "OrderID", Orders, "OrderID" ),
    Columns =                      , Joined,
                    {"OrderDate", "OrderID", "ProductSKU" } ),
    Sorted                  , Columns, "ProductSKU" ),
in  Sorted
```



- Dynamic
  - "M" programs only fail when reaching an invalid evaluation state
  - Static checking, beyond syntax, is an option for tools
- Functional (mostly)
  - Mostly deterministic, no direct side effects, mostly referentially transparent, once calculated, all values are immutable
  - External data is stream processed (not necessarily buffered) and can be non repeatable, error handling can expose non determinism
- Higher-order
  - Functions, closures, and types are also values
  - Nested application and conditionals as only forms of "control flow"
- Optionally typed
  - Mostly optional yet expressive type system, very limited runtime checking of types

- "M" (even recursion) and prefers higher-order application
  - Many library functions take functions as arguments

```
Table SelectRows( table, (row) => row[Manager] = row[Buddy] )
```

primitives ...
Say again?

Examples include constructs
for looping (iteration),

- "M" (even recursion) and prefers higher-order application

  Many library functions take functions as arguments

```
Table SelectRows( table, (row) => row[Manager] = row[Buddy] )
```

Table SelectRows

higher order

takes a function as its argument

primitives ...
Say again?

Examples include constructs
for looping (iteration),

- "M" (even recursion) and prefers higher-order application

  Many library functions take functions as arguments

```
Table.SelectRows( table, (row) => row[Manager] = row[Buddy] )
```

Table SelectRows

higher order

takes a function as its argument

second argument

anonymous

Making the most common

higher-order functions take

table, fields in a record

- "M" (even recursion!) and prefers higher-order application
  - Many library functions take functions as arguments

```
Table.SelectRows( table, (row) => row[Manager] = row[Buddy] )
```

- Often, those parameter functions are unary
  - A special syntactic form helps construct unary function values

```
Table.SelectRows( table, each _[Manager] = _[Buddy] )
```

- An 'each' expression is just shorthand for a unary function
  - The single parameter of an 'each' function is named
- For conciseness, the _ can be omitted when accessing fields or columns (this is the *only* case of syntactic finesse in M)

```
Table.SelectRows( table, each [Manager] = [Buddy] )
```

Evaluation Model

The evaluation model of a language determines how expressions are evaluated.

This can be seen as a refinement of the [illegible]

- Expressions evaluate to values in a context
  - The context binds names to values
- Function application is strict

  No Exceptions style

  "M" has an if expression (the only admission to control flow)

  `if` *condition* `then` *true expression* `else` *false expression*
- Evaluation is eager except for value construction
  - Can deal with infinite lists and tables
  - Can deal with partial records, lists, and tables
    (values containing embedded errors only show when accessed)
- Evaluation 'fails fast' on hard errors
  - Simple model to raise and handle soft errors within M

Going Places

- The model of M combines a very simple (but not trivial) language with a uniform library of functions
  - Adding support for a new domain entails adding functions
- Programming by-example
  - A tool like Power Query "knows" how to generate M for some subset of functions in the library

Power BI

- SharePoint Online – Scheduled Refresh
  - Uploaded Excel workbook that uses Power Query connections
  - Refresh can draw on on-premise data sources through Hybrid Data Delivery (combined Hybrid Proxy in cloud and Informatica Management Gateway server on premise)
  - Gateway runs Mashup provider
- Ongoing work
  - Today, SQL and Oracle only
  - Soon, most PQ sources (but OAuth ones), through gateway
  - Later, all sources and tenant splitting for cloud sources

- Larger declarative forms often require embedded mini languages for expressions
  - M itself is a sma l anguage and the library can be ta lored
- Guaranteed safe semantics are often a concern to avo.d the cost of executing in a container
  - M s safe except for resource exhaustion (stack overflow, heap overflow)
  - M engine can be parameterized to bound stack depth (not in product bits today)
  - M l brary can be subset to control memory pressure

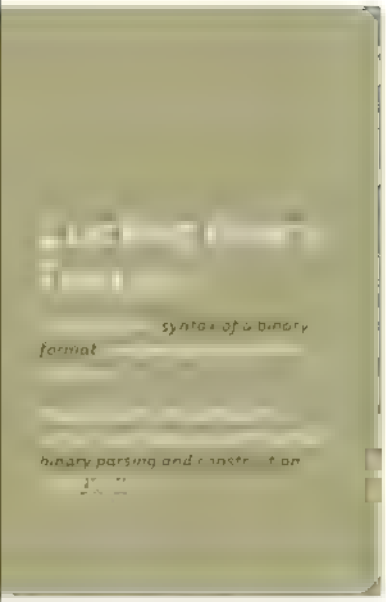
- Data ingress pipelines in Cosmos, Hadoop, etc
  - Use M scripts to describe binary formats
  - Use generic, M-driven, extractor/deserializer to crack wide range of binary formats using such M scripts
- Work in progress
  - M library to describe and crack many kinds of binary formats shipped
  - Generic extractors and deserializers to come


```
fileContents = #binary(
    {0x00, 0x00, 0x00, 0x02,
     0x00, 0x03, 0x00, 0x04,
     0x00, 0x05, 0x00, 0x06}),
```

```
Sample file, viewed in hex
, number of points (2
, point (x: 3, y: 4)
  point (x: 5, y: 6)
```

BinaryFormat functions

- `BinaryFormat.Choice(binaryFormat as function, choice as function, optional combine as function, optional type as nullable type) as function`
  - The binary format that will be used to read the value
  - Choice for the next binary format
  - The function to combine the first value with the second value that was read
  - The type of binary format that will be returned by the choice function, type any, type list, or type binary may be specified
    - For types list or binary, may return a streaming instead of a buffered value, which may reduce memory necessary to read format
- The binary format value produced is processed in five stages
  1. The specified `binaryFormat` is used to read a value
  2. The value is passed to the `choice` function
  3. The `choice` function inspects the value and returns a second binary format
  4. The second binary format is used to read a second value
  5. The second value is returned




```
let
  fileContents = #binary(
      {0x00, 0x00, 0x00, 0x02,
       0x00, 0x03, 0x00, 0x04,
       0x00, 0x05, 0x00, 0x06}),

  pointFormat =                    ([
     x =
     y =
  ]),

  fileFormat =                     (
                                   ,
      (count) =>                   (pointFormat, count))
in
    fileFormat(fileContents)
```

Resulting M value, a list of records

```
{
   [x = 3, y = 4],
   [x = 5, y = 6]
}
```

BinaryFormat functions

- `BinaryFormat.Choice(binaryFormat as function, choice as function, optional combine as function, optional type as nullable type) as function`
  - The binary format that will be used to read the value
  - Choice for the next binary format
  - The function to combine the first value with the second value that was read
  - The type of binary format that will be returned by the choice function, type any, type list, or type binary may be specified
    - For types list or binary, may return a streaming instead of a buffered value, which may reduce memory necessary to read format
- The binary format value produced is processed in five stages
  1. The specified `binaryFormat` is used to read a value
  2. The value is passed to the `choice` function
  3. The `choice` function inspects the value and returns a second binary format
  4. The second binary format is used to read a second value
  5. The second value is returned

- Read atom of specified size and type
  - `Byte`, `SignedInteger16`, `UnsignedInteger16`, `SignedInteger32`, `UnsignedInteger32`, `SignedInteger64`, `UnsignedInteger64`, `Single`, `Double`, `Decimal`, `7BitEncodedSignedInteger`, `7BitEncodedUnsignedInteger`, `Null`
- Create function that reads format
  - `ByteOrder`, `Binary`, `Text`, `List`, `Record`, `Length`, `Choice`, **`Group`**, **`Transform`**
- The execution model (from interpreter of captured expression tree to generated native state machine) is an implementation detail
  - Can be tuned and improved over time
- This small library has been successfully used to handle
  - "Big data" formats like Avro, Ray, ProtoBuf, or Bond
  - Metadata formats embedded in JPEG, PNG, MP3, etc

Double Folding

M is useful both on the "outside" to express most general expressions and on the "inside" to help with point tasks, such as the cracking of binary formats

- M scripts over massive scale-out systems like Cosmos run by folding – from M to Scope in the case of Cosmos
  - Within Scope, a generic M extractor can be used to crack binary streams
- Can fold M to Scope
  - The generated Scope script contains references to the generic M extractor and the extractor's parameterization through given M
- This is a form of "double folding"
  - The folding part of the initial M script is split into an outer Scope script and one or more inner M extractor scripts

Still need a driver, anyone?

Elevators and washing machines have an interesting thing in common: they no longer require a human operator.

And, yes, Google invented the self-driving car. Not.

Power Query Data Sources

This list is continuously growing.

- Web page
- Excel or CSV/PSV/... file
- XML file, JSON file
- Text file
- Folder
- SQL Server database
- Windows Azure SQL database
- Access database
- Oracle database
- IBM DB2 database
- Sybase database
- Teradata database
- MySQL database
- PostgreSQL database
- SharePoint list
- OData feed
- Azure blob and table store
- Hadoop Distributed File System (HDFS)
- Windows Azure HDInsight (Azure Blob Store mapping of HDFS)
- Windows Azure Marketplace feeds and services
- Active Directory
- Facebook graphs
- Exchange
- SAP Business Objects
  - in public preview as of today

Resources

- Power Query has shipped in two versions
  - Standalone (v1) shipped in July 2013
  - Corporate (v2) shipped in February 2014
    - Integrated part of Power BI offering, a subscription service aligned with Office 365
    - http://powerbi.com/
- Tutorials, samples, M language, and M library references
  - http://office.microsoft.com/en-us/excel-help/microsoft-power-query-for-excel-help-HA104003813.aspx